Optimisation:

De l'estimation paramétrique à l'apprentissage, une ballade entre théorie et pratique

S. Delprat

Chapitre 3 – Optimisation avec contrainte

1) Méthode des pénalités

2) Programmation Quadratique Sequentielle *(SQP Sequential Quadratic Programming)*

- *Avec uniquement des contraintes égalités*
- *Avec des contraintes égalités & inégalités*

Méthode des pénalités

On cherche à résoudre le problème suivant:

Critère: $x^* = \arg\min f(x)$ $\qquad x \in \mathbb{R}^n$

Sous les contraintes: $\qquad f \in C^2 : \mathbb{R}^n \to \mathbb{R}$

Egalités $h(x) = 0$ $\qquad h \in C^2 : \mathbb{R}^n \to \mathbb{R}^m$

Inégalités $c(x) \leq 0$ $\qquad c \in C^2 : \mathbb{R}^n \to \mathbb{R}^p$

On suppose que les hypothèses de qualifications sont vérifiées (cf. 1er cours)

On suppose également que le problème est convexe :

- La fonction f est convexe
- Les contraintes définissent un sous-ensemble convexe de $\mathbb{R}^{nx}$

La résolution exacte de ces équations est en général impossible

Méthode des pénalités

Critère: $x^* = \arg\min f(x)$

Sous les contraintes:

Inégalités $c(x) \leq 0$

Egalité $h(x) = 0$

Utilisation de pénalités →

Problème non contraint

$$x^* \approx \arg\min\left[f(x) + P(x,\rho)\right]$$

$$\rho \to 0^+$$

$$P(x,\rho) = \rho\left(\sum_{i=1}^{p} W\left(c_i(x)\right) + \sum_{i=1}^{m}\left(h_i(x)\right)^2\right)$$

<u>Pénalité intérieure (barrières):</u>

Par exemple : $W(x) = -x^{-n}$

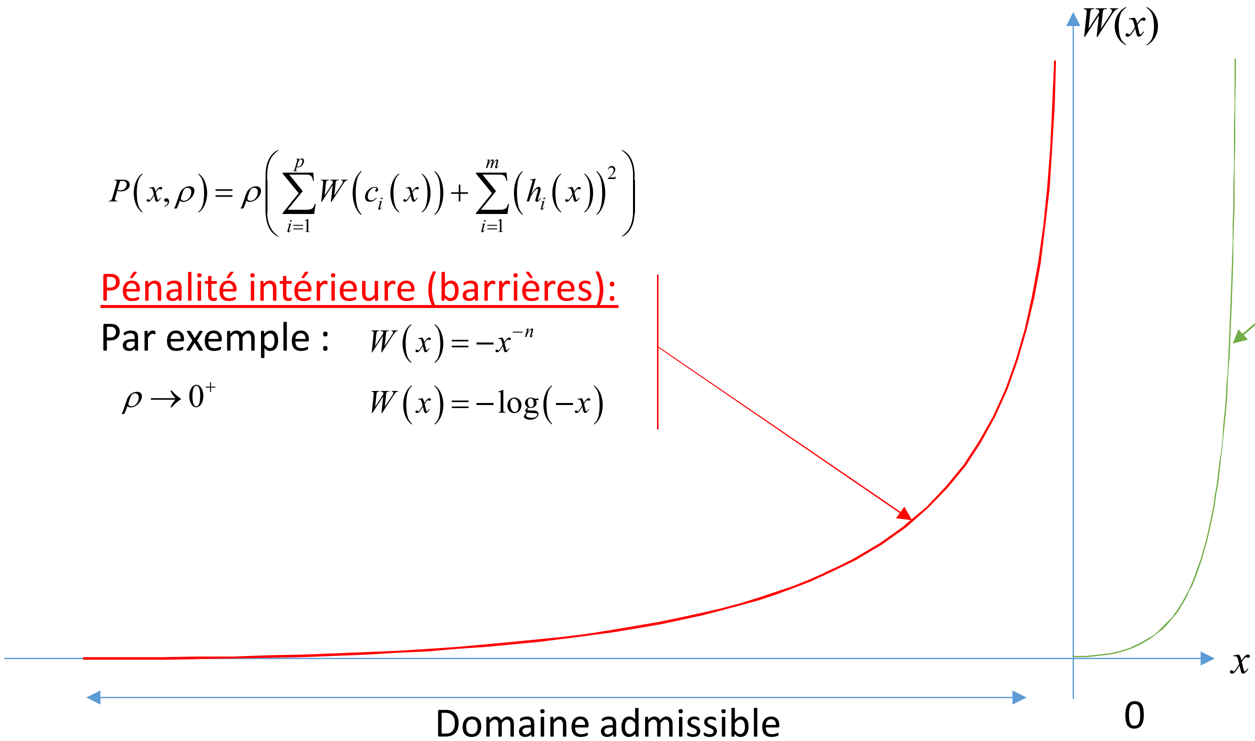
$\rho \to 0^+$ $\qquad W(x) = -\log(-x)$

$$P(x) = \frac{1}{\rho}\left(\sum_{i=1}^{p} W\left(c_i(x)\right) + \sum_{i=1}^{m} W\left(h_i(x)\right)^2\right)$$

<u>Pénalité extérieure:</u>

$W(x) = \left(\max(x,0)\right)^\alpha \qquad \alpha = 1 \text{ ou } 2$

$\rho \to 0^+$



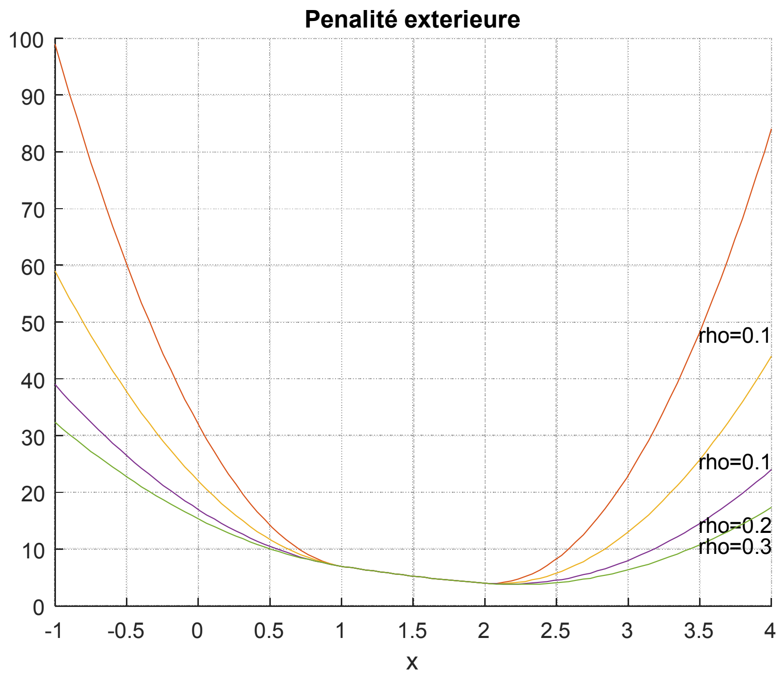
Méthode des pénalités – *pénalité extérieure*

$$\min f(x) = (x-3)^2 + 3$$

$$1 \leq x \leq 2 \Leftrightarrow \begin{cases} 1 - x \leq 0 \\ 2 - x \leq 0 \end{cases}$$

$$W(x) = (\max(0, x))^2$$

```
clear all;
close all;
clc;

% Illustration des fonctions de pénalités
f=@(x) (x-3).^2+3;
c1=@(x) 1-x; %x<1
c2=@(x) x-2; % x<2
W=@(x) max(x,0).^2;

TabRho=[0 0.05 0.1 0.2 0.3];


figure;hold on;
x=linspace(-1,4,100);
for i=1:length(TabRho)
    rho=TabRho(i);
    v=@(x) f(x)+1/rho*(W(c1(x))+W(c2(x)));
    y=v(x);

    plot(x,y);
    grid on
    k=length(x)*0.9;
    text(x(k),y(k)+0.1,sprintf('rho=%.f',rho));
end
title('Penalité exterieure');
xlabel('x');
```



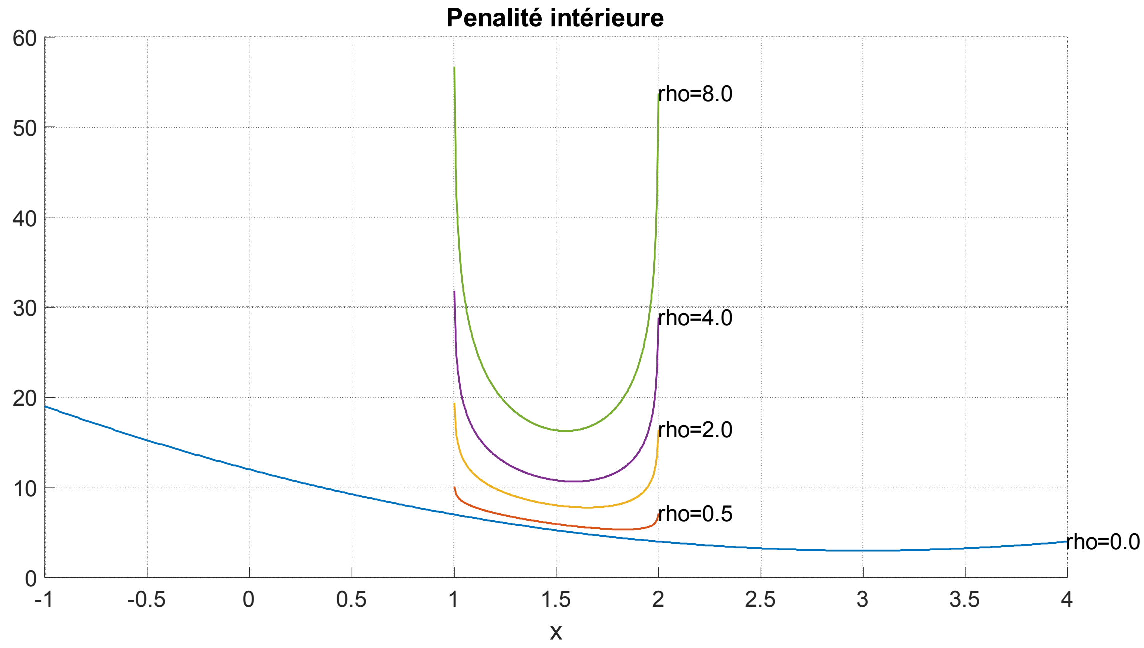
Méthode des pénalités – *pénalité intérieure*

$$\min f(x) = (x-3)^2 + 3$$

$$1 \le x \le 2 \Leftrightarrow \begin{cases} 1 - x \le 0 \\ 2 - x \le 0 \end{cases}$$

$$W(x) = -\ln(-x)$$

```
clear all;
close all;
clc;

% Illustration des fonctions de pénalités
f=@(x) (x-3).^2+3;
c1=@(x) 1-x; %x<1
c2=@(x) x-2; % x<2
W=@(x) -log(-x);

TabRho=[0 0.5 2 4 8];


figure;hold on;
for i=1:length(TabRho)
    rho=TabRho(i);
    if rho==0
        x=linspace(-1,4,500);
    else
        x=linspace(1,2,500);
    end
    v=@(x) f(x)+rho*(W(c1(x))+W(c2(x)));
    y=v(x);

    plot(x,y);
    grid on
    text(x(end-1),y(end-1)+0.1,sprintf('rho=%.1f',rho));
end
title('Penalité intérieure');
xlabel('x');
```



<u>Attention:</u> avec les pénalités intérieures, il faut garantir que l'algorithme génère des solutions intermédiaires à l'intérieure du domaine

Méthode des pénalités - *exemple*

Problème contraint: $\min g(x,y) = e^{x-3y-\frac{1}{10}} + e^{x+3y-\frac{1}{10}} + e^{-x-\frac{1}{10}}$

$$x > 0$$

$$y > 0.5$$

Par exemple, on utilise une pénalité externe:

$$x > 0 \Leftrightarrow c_1(x,y) < 0 \qquad c_1(x,y) = -x$$

$$y > 0.5 \Leftrightarrow c_2(x,y) < 0 \qquad c_2(x,y) = 0.5 - y$$

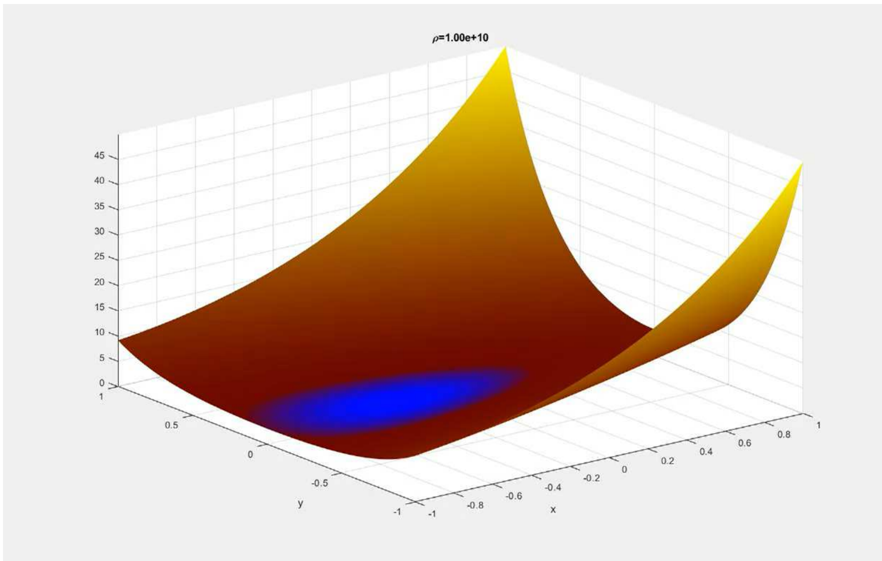
$$\Gamma(z) = \begin{cases} 1 & si \quad z > 0 \\ 0 & \text{sinon} \end{cases}$$

$$W(c_1(x,y)) = (-x)^2 \, \Gamma(c_1(x,y)) \qquad W(c_2(x,y)) = (0.5 - y)^2 \, \Gamma(c_2(x,y))$$

Le problème contraint est ainsi transformé en problème non contraint:

$$f(x,y) = g(x,y) + \frac{1}{\rho}\left[W(c_1(x,y)) + W(c_2(x,y))\right]$$

Animation



Problème non contraint:

$$\min f(x,y) = g(x,y) + \frac{1}{\rho}\left[W(c_1(x,y)) + W(c_2(x,y))\right]$$

$\Rightarrow$ Peut être résolu en utilisant les algos vus au chapitre 2 (Gradient, Newton, Région de confiance, etc.)

Méthode des pénalités - *exemple*

$$f(x,y)=g(x,y)+\frac{1}{\rho}\left[W\left(c_1(x,y)\right)+W\left(c_2(x,y)\right)\right]$$

On utilise une pénalité externe.

$$x>0 \Leftrightarrow c_1(x,y)<0 \qquad c_1(x,y)=-x$$

$$y>0.5 \Leftrightarrow c_2(x,y)<0 \qquad c_2(x,y)=0.5-y$$

$$\Gamma(z)=\begin{cases}1 & si \quad z>0\\ 0 & \text{sinon}\end{cases}$$

$$W\left(c_1(x,y)\right)=(-x)^2\,\Gamma\left(c_1(x,y)\right)$$

$$W\left(c_2(x,y)\right)=(0.5-y)^2\,\Gamma\left(c_2(x,y)\right)$$

Il faut calculer le gradient et le Hessien de la fonction pénalisée:

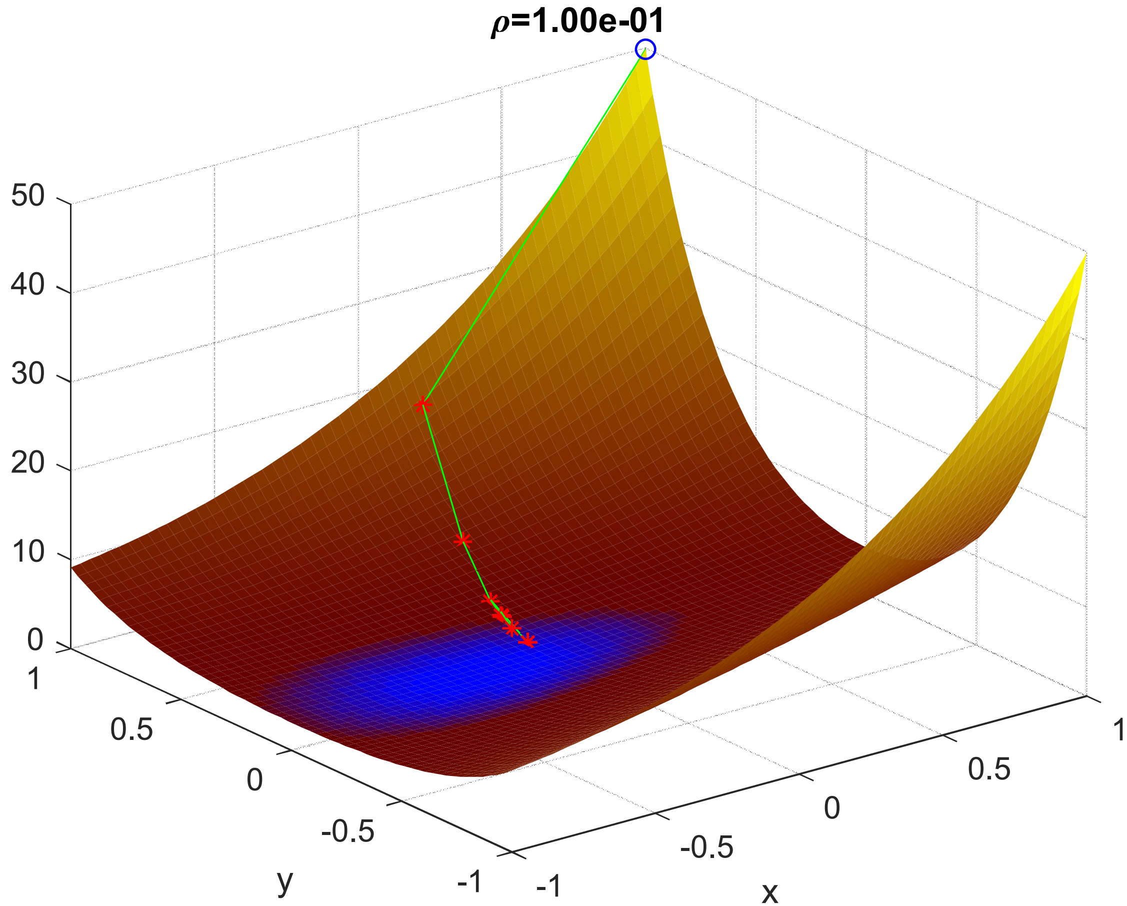
Gradient: $\nabla f(x,y)=\nabla g(x,y)+\frac{1}{\rho}\begin{pmatrix}2x\Gamma\left(c_1(x,y)\right)\\ (2y-1)\Gamma\left(c_2(x,y)\right)\end{pmatrix}$

Hessien $\nabla^2 f(x,y)=\nabla^2 g(x,y)+\frac{1}{\rho}\begin{pmatrix}2\Gamma\left(c_1(x,y)\right) & 0\\ 0 & 2\Gamma\left(c_2(x,y)\right)\end{pmatrix}$

Méthode des pénalités - *exemple*

Dans les programmes non contraints, il suffit de modifier l'expression de la fonction et son gradient:

```
rho=0.1;
n=2

% Pénalité externe
W=@(x) (x>0).*x^2;

% Rajout de contraintes 0<x1  0.5<x2
c1=@(x) 0-x(1); %0<x1
c2=@(x) 0.5-x(2); % 0.5<x2

dP=@(x) [-n*(-x(1))^(n-1)*(c1(x)>0);
         -n*(0.5-x(2))^(n-1)*(c2(x)>0)];

d2P=@(x) [n*(n-1)*(-x(1))^(n-2)*(c1(x)>0) 0;
          0 n*(n-1)*(0.5-x(2))^(n-2)*(c2(x)>0)];

% Fonctions étendues
f=@(x) g(x)+1./rho*(W(c1(x))+W(c2(x)));
df=@(x) dg(x) + 1./rho*dP(x);
d2f=@(x) d2g(x) + 1./rho*d2P(x);
```



ρ=0,1

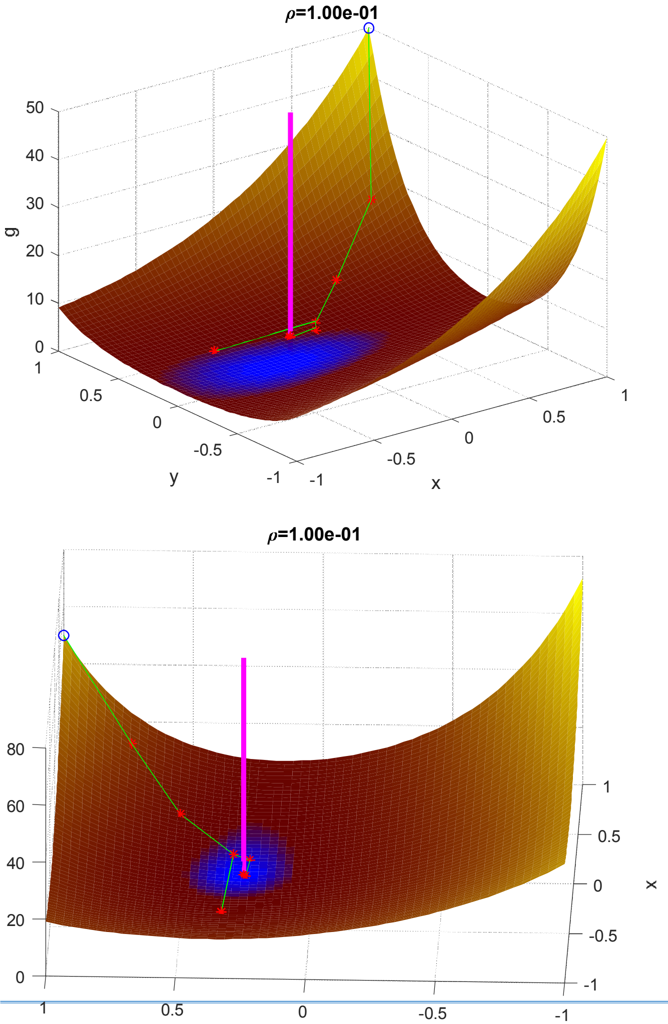
x=[-0.04,0.29]

Trust Region + Dodleg + Penalité
6 itérations boucle principale
50 appels de f

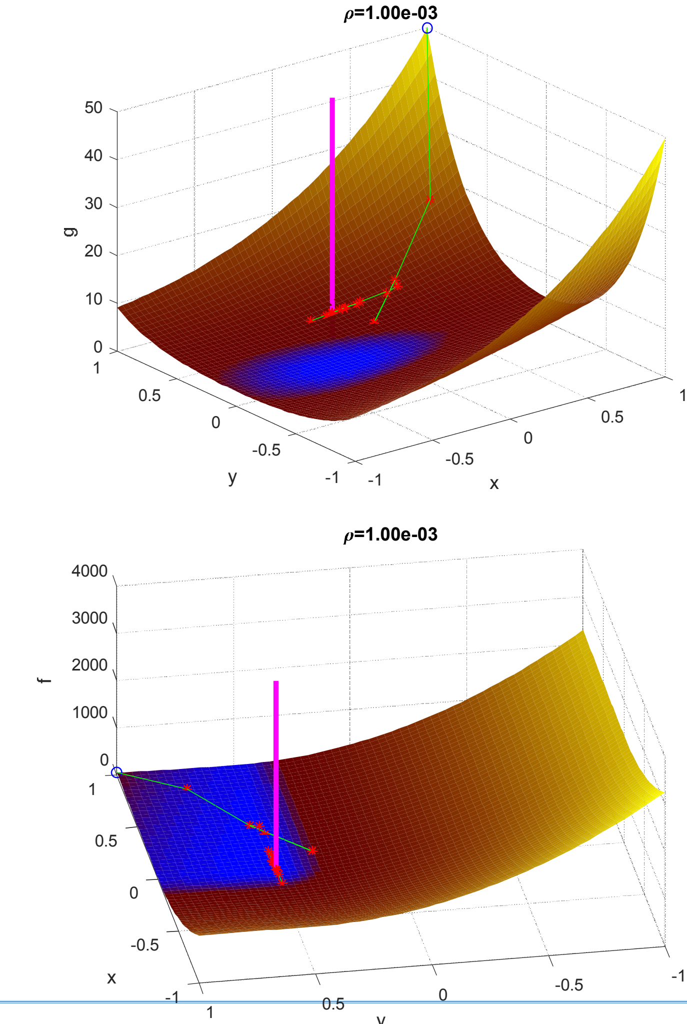
=> Loin de la sol optimale x*=[0,0.5] car ρ est trop grand

Méthode des pénalités - *exemple*

ρ=0,1 x=[-0.07,0.27] 31 appels de f

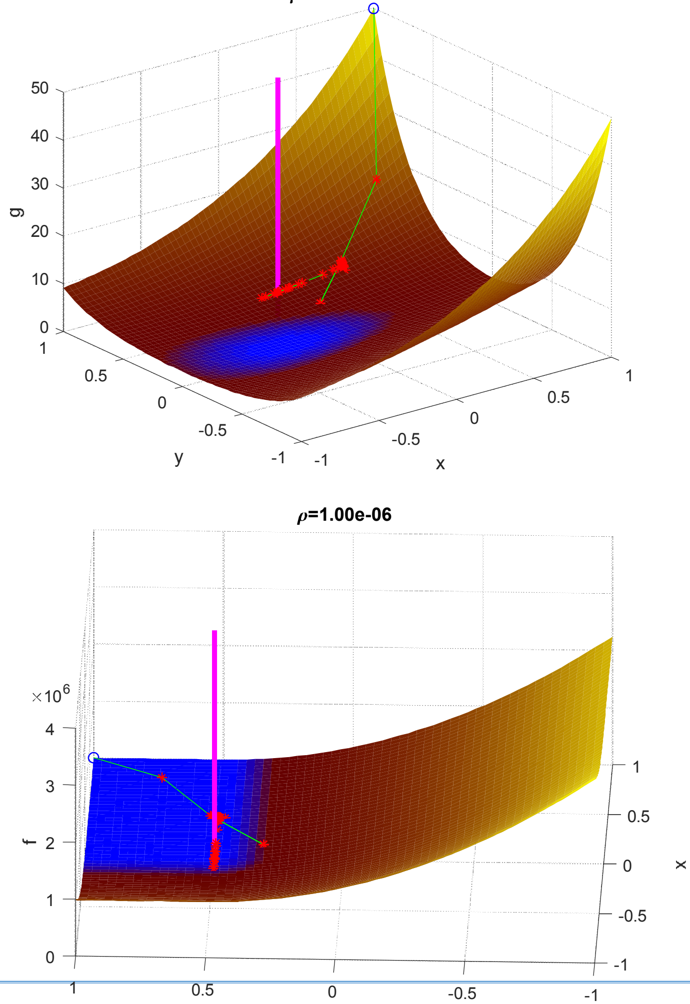


$\rho=10^{-3}$ x=[-0.0016,0.4943] 47 appels de f



$\rho=10^{-6}$ x=[-1.6761e-06, 4.9999e-01]

159 appels de f



Méthode des pénalités - *conclusion*

- Les méthode de pénalités sont simples à mettre en œuvre

=> Il suffit d'utiliser un algorithme non contraint

- Pénalités extérieures: la solution peut se trouver légèrement à l'extérieur du domaine admissible

=> Les fonctions doivent être définies à l'extérieur des contraintes

- Pénalités intérieure: les pénalités ne sont pas définies à l'extérieur du domaine

$\Rightarrow$ Les algorithmes doivent garantir que les solutions intermédiaires restent à l'intérieur du domaine

$\Rightarrow$ Nécessite des modifications

- Trouver une « bonne » valeur du poids ρ n'est pas simple

=> Sur un problème réel de grande taille, il faut vérifier si la solution converge vers une valeur unique lorsque $\rho \to 0+$

- Numériquement, les méthodes de pénalités peuvent poser des problèmes numériques (Hessien mal conditionné)

$$\nabla^2 f(x,y) = \nabla^2 g(x,y) + \begin{pmatrix} \boxed{\dfrac{2\Gamma(c_1(x,y))}{\rho}} & 0 \\ 0 & \boxed{\dfrac{2\Gamma(c_2(x,y))}{\rho}} \end{pmatrix} \qquad \to +\infty$$

1) Méthode des pénalités

2) Programmation Quadratique Sequentielle *(SQP Sequential Quadratic Programming)*

- *Avec uniquement des contraintes égalités*
- *Avec des contraintes égalités & inégalités*

Sequential Quadratic Programming – *conditions KKT*

On considère maintenant le problème suivant avec $\boldsymbol{m}$ **contraintes égalité** :

$f:\mathbb{R}^n \to \mathbb{R}$ $\quad h:\mathbb{R}^n \to \mathbb{R}^m$ toutes de classe $\mathcal{C}^2$.

$$\min_{x\in\mathbb{R}^n} f(x)$$

sous les contraintes: $h(x)=0$

Nb : le cas avec contraintes $c(x)\leq 0$ sera traité ultérieurement

Les méthodes de pénalités ne tiennent pas compte des conditions d'optimalité avec contrainte (KKT)

=> La solution obtenue est une approximation

=> Idée: résoudre les conditions d'optimalité numériquement

Sequential Quadratic Programming – *Rappel : Gradient, Jacobienne, Hessien, etc.*

Fonction scalaire multi-variable à valeur dans $\mathbb{R}$:

$$x = \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix}^T$$

Fonction: $f(x) : \mathbb{R}^n \to \mathbb{R}$

Gradient: $\nabla f(x) : \mathbb{R}^n \to \mathbb{R}^n$

$$\nabla f(x) = \begin{pmatrix} \dfrac{\partial f}{\partial x_1} & \cdots & \dfrac{\partial f}{\partial x_n} \end{pmatrix}^T$$

Hessien: $H_f(x) = \nabla^2 f(x) : \mathbb{R}^n \to \mathbb{R}^{n\times n}$

$$H_f(x) = \nabla^2 f(x) = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Fonction vectorielle multi-variable à valeur dans $\mathbb{R}^m$:

$$x = \begin{pmatrix} x_1 & \cdots & x_n \end{pmatrix}^T$$

Fonction: $f(x) : \mathbb{R}^n \to \mathbb{R}^m$

$f_i(x) : \mathbb{R}^n \to \mathbb{R} \qquad i = 1 \cdots m$

$$f(x) = \begin{pmatrix} f_1(x) & \cdots & f_m(x) \end{pmatrix}^T$$

Jacobienne: $\nabla f = J_f(x) : \mathbb{R}^n \to \mathbb{R}^{m\times n}$

$$\nabla f(x) = J_f(x) = \begin{pmatrix} (\nabla f_1(x))^T \\ \vdots \\ (\nabla f_m(x))^T \end{pmatrix}$$

$$\nabla f(x) = J_f(x) = \begin{pmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \dfrac{\partial f_m}{\partial x_2} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{pmatrix}$$

Sequential Quadratic Programming – *conditions KKT*

Rappel : Conditions de Karush, Kuhn et Tucker :

Lagrangien: $L(x,\lambda) = f(x) + \lambda^T h(x)$

Condition au premier ordre: $\nabla L(x,\lambda) = 0$

$$\nabla L(x,\lambda) = \begin{pmatrix} \dfrac{\partial L(x,\lambda)}{\partial x} \\ \dfrac{\partial L(x,\lambda)}{\partial \lambda} \end{pmatrix} = 0$$

$$\nabla_x L(x,\lambda) = \frac{\partial L(x,\lambda)}{\partial x} = 0 \Leftrightarrow \nabla f(x) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x) = 0$$

$$\Leftrightarrow \nabla f(x) + \left(\nabla h(x)\right)^T \lambda = 0$$

$$\frac{\partial L(x,\lambda)}{\partial \lambda} = 0 \Leftrightarrow h(x) = 0$$

Au final, la condition au premier ordre est:

$$\begin{pmatrix} \nabla f(x) + \left(\nabla h(x)\right)^T \lambda \\ h(x) \end{pmatrix} = \begin{pmatrix} \nabla_x L(x,\lambda) \\ h(x) \end{pmatrix} = 0$$

=> tient compte du critère ET des contraintes

Sequential Quadratic Programming – *conditions KKT*

Conditions d'optimalité de Karush, Kuhn et Tucker , la solution vérifie :

$$\begin{pmatrix} \nabla f(x) + (\nabla h(x))^T \lambda \\ h(x) \end{pmatrix} = 0$$

Il s'agit donc de trouver les racines d'une fonction non linéaire multi variables.

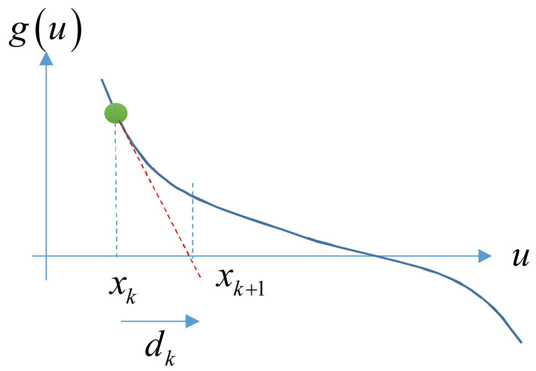
=> de la forme $g(u) = 0$

$$g : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m} \qquad u = \begin{pmatrix} x \\ \lambda \end{pmatrix} \qquad g(u) = \begin{pmatrix} \nabla_x L(x,\lambda) \\ h(x) \end{pmatrix}$$

Rappel :Résolution d'un problème de le forme $g(u)$=0 avec la méthode de Newton

Résoudre: $-g(u^k) = \nabla g(u^k) d^k$

Itérer $x^{k+1} = x^k + d^k$



Sequential Quadratic Programming – *conditions KKT*

Rappel :Résolution d'un problème de le forme $g(u)$=0 par la Méthode de Newton

Résoudre: $-g\left(u^k\right)=\nabla g\left(u^k\right)d^k$

Itérer $x^{k+1}=x^k+d^k$

On a: $g\left(u\right)=\begin{pmatrix}\nabla_x L\left(x,\lambda\right)\\ h\left(x\right)\end{pmatrix}=\begin{pmatrix}\nabla f\left(x\right)+\left(\nabla h\left(x\right)\right)^T\lambda\\ h\left(x\right)\end{pmatrix}=\begin{pmatrix}\nabla f\left(x\right)+\sum_{i=1}^{m}\nabla h_i\left(x\right)\lambda_i\\ h\left(x\right)\end{pmatrix}$

et son gradient $\nabla g\left(u\right)=\begin{pmatrix}\nabla^2 f\left(x\right)+\sum_{i=1}^{m}\nabla^2 h_i\left(x\right)\lambda_i & \left(\nabla h\left(x\right)\right)^T\\ \nabla h\left(x\right) & 0\end{pmatrix}$

On applique la formule de Newton: $-g\left(u^k\right)=\nabla g\left(u^k\right)d^k$

$$-\begin{pmatrix}\nabla_x L\left(x,\lambda\right)\\ h\left(x^k\right)\end{pmatrix}=\begin{pmatrix}\nabla^2 f\left(x^k\right)+\sum_{i=1}^{m}\nabla^2 h_i\left(x^k\right)\lambda_i^k & \left(\nabla h\left(x^k\right)\right)^T\\ \nabla h\left(x^k\right) & 0\end{pmatrix}\begin{pmatrix}d_x^k\\ d_\lambda^k\end{pmatrix}$$

Sequential Quadratic Programming – *conditions KKT*

<u>Résolution d'un problème de le forme $g(x)$=0 : Méthode de Newton</u>

On pose: $H^k = \nabla^2 f\left(x^k\right) + \sum_{i=1}^{m} \nabla^2 h_i\left(x^k\right)\lambda_i^k$

Itération de Newton:

$$\begin{pmatrix} \nabla_x L(x,\lambda) \\ h\left(x^k\right) \end{pmatrix} = -\underbrace{\begin{pmatrix} H^k & \left(\nabla h\left(x^k\right)\right)^T \\ \nabla h\left(x^k\right) & 0 \end{pmatrix}}_{\text{Supposée inversible : qualification des contraintes+ le Hessien de f défini positif}} \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix}$$

<u>Essayons de comprendre à quoi tout cela corresponds?</u>

Ré-écriture:

$$\begin{cases} \nabla_x L(x,\lambda) = -H^k d_x^k - \left(\nabla h\left(x^k\right)\right)^T d_\lambda^k \\ h\left(x^k\right) = -\nabla h\left(x^k\right) d_x^k \end{cases}$$

$$\nabla_x L(x,\lambda) = \nabla f(x) + \left(\nabla h\left(x^k\right)\right)^T \lambda^k$$

$$\begin{cases} \nabla f(x) + \left(\nabla h\left(x^k\right)\right)^T \lambda^k = -H^k d_x^k - \left(\nabla h\left(x^k\right)\right)^T d_\lambda^k \\ h\left(x^k\right) + \nabla h\left(x^k\right) d_x^k = 0 \end{cases}$$

Sequential Quadratic Programming – *conditions KKT*

<u>Résolution d'un problème de le forme $g(x)$=0 : Méthode de Newton</u>

$$\begin{cases} \nabla f(x) = -H^k d_x^k - \left(\nabla h\left(x^k\right)\right)^T \left(d_\lambda^k + \lambda^k\right) \\ h\left(x^k\right) - \nabla h\left(x^k\right) d_x^k = 0 \end{cases}$$

$$\lambda^{k+1} = d_\lambda^k + \lambda^k$$

$$\begin{cases} H^k d_x^k + \nabla f(x) = \left(\nabla h\left(x^k\right)\right)^T \lambda^{k+1} \\ h\left(x^k\right) - \nabla h\left(x^k\right) d_x^k = 0 \end{cases}$$

=> Corresponds au conditions KKT du problème

$$\min J = \left(d_x^k\right)^T H^k d_x^k + \nabla f(x) d_x^k$$

Approximation quadratique du critère $f(x)$

$$h\left(x^k\right) + \nabla h\left(x^k\right) d_x^k = 0$$

Approximation linéaire des contraintes $h(x)$

Sequential Quadratic Programming – *approximation quadratique*

$$\min f(x,y) = e^{x-3y-\frac{1}{10}} + e^{x+3y-\frac{1}{10}} + e^{-x-\frac{1}{10}}$$

$$c(x,y) = (y-0.5)^2 - x = 0$$

$$\min J = \left(d_x^k\right)^T H^k d_x^k + \nabla f(x) d_x^k$$

$$h\left(x^k\right) + \nabla h\left(x^k\right) d_x^k = 0$$

<ins>Visualisation des approximations du critère + contrainte:</ins>

Point de linéarisation

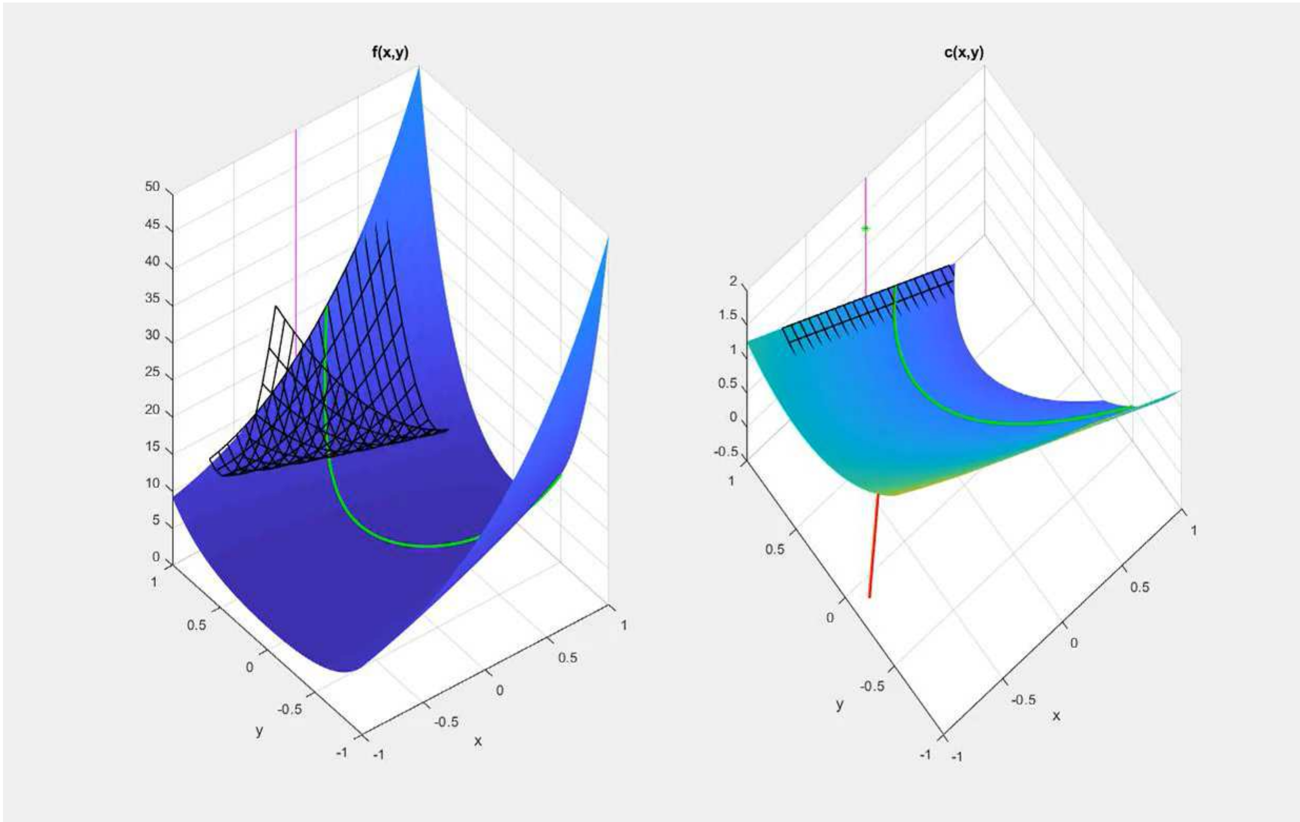
Approximations:

- Quadratique du critère
- Linéaire de la contrainte

Courbe de niveau $c(x)=0$

Courbe de niveau $h\left(x^k\right) + \nabla h\left(x^k\right) d_x^k = 0$

(approx. Linéaire de la contrainte)



Résumé :

On cherche (d_x^k, d_λ^k) solution de l'itération de Newton, c'est-à-dire la résolution du problème local:

$$-\begin{pmatrix} \nabla_x L(x,\lambda) \\ h(x^k) \end{pmatrix} = \begin{pmatrix} \nabla^2 f(x^k) + \sum_{i=1}^{m} \nabla^2 h_i(x^k)\lambda_i^k & \left(\nabla h(x^k)\right)^T \\ \nabla h(x^k) & 0 \end{pmatrix} \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix}$$

On itère ensuite depuis le point précédent: $\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} + \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix}$

Question : Comment choisir le point initial $\begin{pmatrix} x^0 \\ \lambda^0 \end{pmatrix}$?

Sequential Quadratic Programming – *initialisation de λ^0*

Question : Comment choisir le point initial $\begin{pmatrix} x^0 \\ \lambda^0 \end{pmatrix}$?

On utilise l'approximation quadratique du Lagrangien:

Conditions KKT au point initial : $\begin{pmatrix} \nabla f(x^0) + \left(\nabla h(x^0)\right)^T \lambda^0 \\ h(x^0) \end{pmatrix} = 0$

$$\Leftrightarrow \nabla f(x^0) + \left(\nabla h(x^0)\right)^T \lambda^0 = 0$$

$$\Leftrightarrow \left(\nabla h(x^0)\right)^T \lambda^0 = -\nabla f(x^0)$$

$$\Leftrightarrow \left(\nabla h(x^0)\right)\left(\nabla h(x^0)\right)^T \lambda^0 = -\left(\nabla h(x^0)\right)\nabla f(x^0)$$

D'où l'approximation: $\lambda^0 = -\left(\left(\nabla h(x^0)\right)\left(\nabla h(x^0)\right)^T\right)^{-1}\left(\nabla h(x^0)\right)\nabla f(x^0)$

Sequential Quadratic Programming – *algorithme de base*

Paramètres x^0

Algorithme

$$k = 0$$

$$\lambda^0 = -\left(\left(\nabla h\left(x^0\right)\right)\left(\nabla h\left(x^0\right)\right)^T\right)^{-1}\left(\nabla h\left(x^0\right)\right)\nabla f\left(x^0\right)$$

Tant que critère non satisfait:

1) Calculer $\left(d_x^k, d_\lambda^k\right)$ solution de $\begin{pmatrix} \nabla^2 f\left(x^k\right) + \sum_{i=1}^{m} \nabla^2 h_i\left(x^k\right)\lambda_i^k & \left(\nabla h\left(x^k\right)\right)^T \\ \nabla h\left(x^k\right) & 0 \end{pmatrix}\begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x L(x,\lambda) \\ h\left(x^k\right) \end{pmatrix}$

2) $\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} = \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} + \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix}$

3) $k = k + 1$

Fin Tant que

Sequential Quadratic Programming – *algorithme de base*

Résolution du système d'équations linéaire
$$\begin{pmatrix} \nabla^2 f\left(x^k\right)+\sum_{i=1}^{m}\nabla^2 h_i\left(x^k\right)\lambda_i^k & \left(\nabla h\left(x^k\right)\right)^T \\ \nabla h\left(x^k\right) & 0 \end{pmatrix}\begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x L(x,\lambda) \\ h\left(x^k\right) \end{pmatrix}$$

1) En pratique il faut éviter d'inverser des Matrices : pb. numérique, long, etc.

=> Heureusement, de nombreux algorithmes numériques existent et utilisent des factorisations particulières

=> Non traité dans ce cours, voir un cours de math appli et/ou optimisation

2) Le hessien $\nabla^2 f\left(x^k\right)$ n'est pas disponible et peut ne pas être défini positif (problème non convexe)

=> Utilisation d'une méthode « Quasi Newton » de type DFP ou BFGS

$$x^{k+1} = x^k + d^k$$

$$\gamma^k = \nabla f\left(x^{k+1}\right) - \nabla f\left(x^k\right)$$

$$\delta^k = d^k$$

$$S^{k+1} = \left(I - \frac{\delta^k\left(\gamma^k\right)^T}{\left(\delta^k\right)^T\gamma^k}\right)S^k\left(I - \frac{\delta^k\left(\gamma^k\right)^T}{\left(\delta^k\right)^T\gamma^k}\right) + \frac{\delta^k\left(\delta^k\right)^T}{\left(\delta^k\right)^T\gamma^k}$$

Génération automatique de fonction pour le critère et la contraintes ainsi que leur dérivées et jacobienne:

Savoir-faire n°1 : Programmation Matlab

`Savoir générer automatiquement le code Matlab d’une expression symbolique`

Syntaxe: `matlabFunction(Expression,’File’,NomFich,’Vars’,[x;y]);`

- ⇒ Génère la fonction matlab correspondant au code de l’`Expression`
- ⇒ dans le fichier `NomFich`
- ⇒ Le paramètre de la fonction Matlab est un vecteur colonne contenant `x` et `y`

L’instruction matlabFunction permet également de génrer une fonction inline, etc. => cf doc.

```
clear all;
close all;
clc

syms x y z
f(x,y,z)= x^2+(x-y)^2+z;
```

```
g=matlabFunction(f);
```

Une fonction en mémoire :

```
g =
 function_handle with value:
   @(x,y,z)z+(x-y).^2+x.^2
```

```
matlabFunction(f,'file','demo1','vars',{x,y,z})
```

Une fonction (fichier) avec 3 entrées x,y,z

```
matlabFunction(f,'file','demo2','vars',{x,[y,z]})
```

Une fonction (fichier) avec 2 entrées : 1 variable x et un vecteur (y,z):

Sequential Quadratic Programming – *algorithme de base*

Problème étudié:

$$\begin{cases} \min f(x,y) = (1-x)^2 + 100(y-x^2)^2 \\ h(x,y) = (y-0.5)^2 - x = 0 \end{cases}$$

Génération automatique de fonction pour le critère et la contraintes ainsi que leur dérivées et jacobienne:

```
clear all
close all
clc

syms x y

% Critère
f(x,y)= (1-x).^2+100*(y-x^2)^2;
df=gradient(f);
d2f=hessian(f);

% c
h=(y-0.5).^2-x;
Jh=jacobian(h,[x;y]);
d2h=hessian(h,[x;y]);

matlabFunction(f,'File','f','Vars',{[x; y]});
matlabFunction(df,'File','df','Vars',{[x; y]});
matlabFunction(d2f,'File','d2f','Vars',{[x; y]});
matlabFunction(h,'File','h','Vars',{[x;y]});
matlabFunction(Jh,'File','Jh','Vars',{[x;y]});
matlabFunction(d2h,'File','d2h','Vars',{[x;y]});
```

```
function f = f(in1)
%F
%    F = F(IN1)

%    This function was generated by the Symbolic Math Toolbox version 8.0.
%    31-Oct-2017 13:03:00

x = in1(:,1);
y = in1(:,2);
t2 = x-1.0;
t3 = y-x.^2;
f = t2.^2+t3.^2.*1.0e2;
```

```
function d2f = d2f(in1)
%D2F
%    D2F = D2F(IN1)

%    This function was generated by the Symbolic Math Toolbox version 8.0.
%    31-Oct-2017 13:03:00

x = in1(:,1);
y = in1(:,2);
d2f = reshape([y.*-4.0e2+x.^2.*1.2e3+2.0,x.*-4.0e2,x.*-4.0e2,2.0e2],[2,2]);
```

Sequential Quadratic Programming – *algorithme de base*

```
clear all;
close all;
clc;


% Condition initiale
x0=[0.5;-1];

% Zone d'intéret pour les graphiques
xmin=-1; xmax=1; ymin=-1; ymax=1;

% Affichage de la fonction
X=linspace(xmin,xmax,61);
Y=linspace(ymin,ymax,62);
Z=zeros(length(Y),length(X));
Z2=Z;
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f([X(i);Y(j)]);
    end
end
Ycontrainte=-0.5:0.1:1;
Xcontrainte=(Ycontrainte-0.5).^2;
for i=1:length(Ycontrainte)
    Zcontrainte(i)=f([Xcontrainte(i);Ycontrainte(i)]);
end

figure(10);
surf(X,Y,Z,'EdgeColor','none');
hold on;
xlabel('x');
ylabel('y');
hold on;
plot3(x0(1),x0(2),f(x0),'bo');
plot3(Xcontrainte,Ycontrainte,Zcontrainte,'r','linewidth',2)
xlabel('x');
ylabel('y');
zlabel('g');
```

```
% Criteres d'arret
Tolf=1e-8;
Tolx=1e-9;
IterMax=100;

% Estimation du paramètre de Lagrange initial
gf=df(x0);
gh=Jh(x0);
l0=-(gh*gh')^-1*gh*gf;

% Algorithme SQP contrainte égalité
k=1;
x=x0;
l=l0;
ended=0;
n=size(x0,1);
while ended==0
    xk=x(:,k);
    fxk=f(xk);
    Jhxk=Jh(x(:,k));
    dfx=df(x(:,k));

    H=d2f(x(:,k))+d2h(x(:,k))*l(k);
    dLx=dfx+Jhxk'*l(k);

    A=[H Jhxk';
        Jhxk 0];
    B=-[dLx;h(x(:,k))];
    Phi=A\B;


    dx=Phi(1:n);
    dl=Phi(n+1);
    x(:,k+1)=x(:,k)+dx;
    l(k+1)=l(k)+dl;
```

Pas très efficace, peut être amélioré

Sequential Quadratic Programming – *algorithme de base*

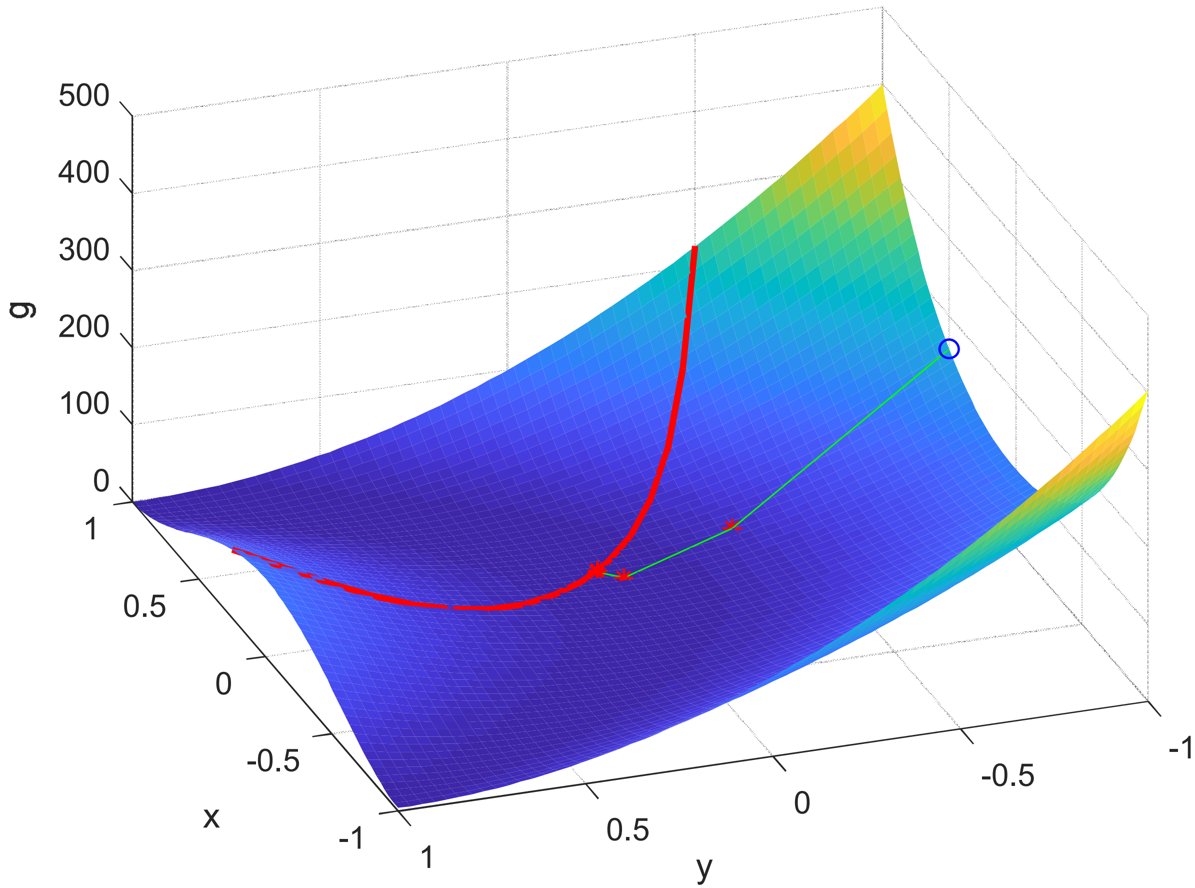
```
    % Affichage
    xkp1=x(:,k+1);

    figure(10)
    plot3(xkp1(1),xkp1(2),f(xkp1),'r*');
    plot3([xkp1(1) xk(1)],[xkp1(2) xk(2)],[f(xkp1) fxk],'g');


    fxkp1=f(x(:,k+1))+l(k)*h(x(:,k+1));
    ended=(k>IterMax)||...
        (norm(fxkp1-fxk)<Tolf)||...
        (norm([x(:,k)-x(:,k+1);l(k)-l(k+1)])<Tolx);
    k=k+1;

end

fprintf('Valeur finale : x=[%.2f,%.2f] \n',x(1,k),x(2,k));
fprintf('Nombre d''iterations : %i\n',k)
if k>IterMax
    fprintf(' Iteration maximale atteinte\n');
end
if norm(fxkp1-fxk)<Tolf
    fprintf('Plus d''amélioration de f\n')
end
k=k-1;
if norm(x(:,k)-x(:,k+1))<Tolx
    fprintf('plus d''amélioration de x\n');
end
```

Sequential Quadratic Programming – *approximation quadratique*

$$\begin{cases}\min f(x,y)=(1-x)^2+100(y-x^2)^2\\ h(x,y)=(y-0.5)^2-x=0\end{cases}$$

x=[0.21,0.04]

SQP 1 contrainte égalité
6 itérations boucle principale



1) Méthode des pénalités

2) Programmation Quadratique Sequentielle *(SQP Sequential Quadratic Programming)*

- *Avec uniquement des contraintes égalités*
- *Avec des contraintes égalités & inégalités*

Sequential Quadratic Programming – *contraintes égalités & inégalités*

On considère maintenant le problème suivant avec m contraintes égalité et p contraintes inégalités :

$f:\mathbb{R}^n \to \mathbb{R} \qquad h:\mathbb{R}^n \to \mathbb{R}^m \qquad c:\mathbb{R}^n \to \mathbb{R}^p$ toutes de classe $\mathcal{C}^2$.

$$\min_{x\in\mathbb{R}^n} f(x)$$

sous les contraintes: $h(x)=0$

$c(x)\leq 0$

Une vrai fausse bonne idée :

Convertir les contraintes inégalités en contraintes égalités:

$$c(x)\leq 0 \Leftrightarrow c_i(x)+s_i^2=0 \qquad i=1..p$$

Théoriquement, cela doit fonctionner.
En pratique, engendre des algorithmes numériquement instables
=> A éviter

Sequential Quadratic Programming – *Active set method*

On cherche à résoudre le problème suivant:

Critère: $\min f(x)$ $\quad x \in \mathbb{R}^{nx} \quad f \in C^2 : \mathbb{R}^{nx} \to \mathbb{R}$

Sous les contraintes: $c(x) \leq 0$ $\quad c \in C^2 : \mathbb{R}^{nx} \to \mathbb{R}^{nc}$

Conditions KKT:

Lagrangien: $L(x,\mu,\lambda) = f(x) + \mu^T c(x)$

Stationnarité: $\nabla L(x,\mu,\lambda) = 0 \Leftrightarrow \nabla f(x^*) + \sum_{j=0}^{p} \mu_j^* \nabla c_j(x^*)$

$$c(x^*) \leq 0$$

$$\mu_i^* \geq 0$$

$$\mu_i^* c_i(x^*) = 0$$

<u>Contrainte est active</u>

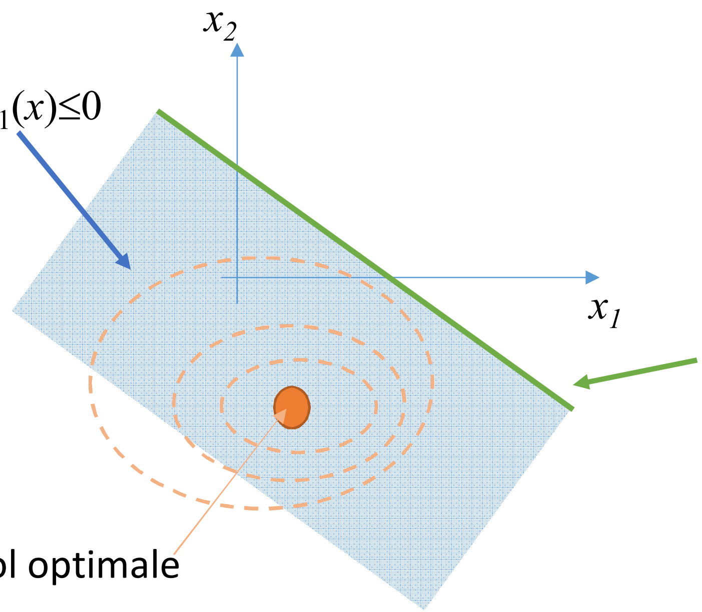
- La solution est sur la frontière
- le multiplicateur est positif

<u>Contrainte inactive</u>

- La solution est à l'intérieur du domaine admissible
- Le multiplicateur est nul

Sequential Quadratic Programming – *Active set method*

Exemple de contrainte inégalité: $c_1(x) = y + x \le 0 \Leftrightarrow y \le -x$



Cas n°1 : solution dans le domaine admissible

La contrainte ne change rien

$$c(x) < 0 \qquad \mu = 0$$

$$L(x,\mu,\lambda) = f(x) + \mu^T c(x) = f(x)$$

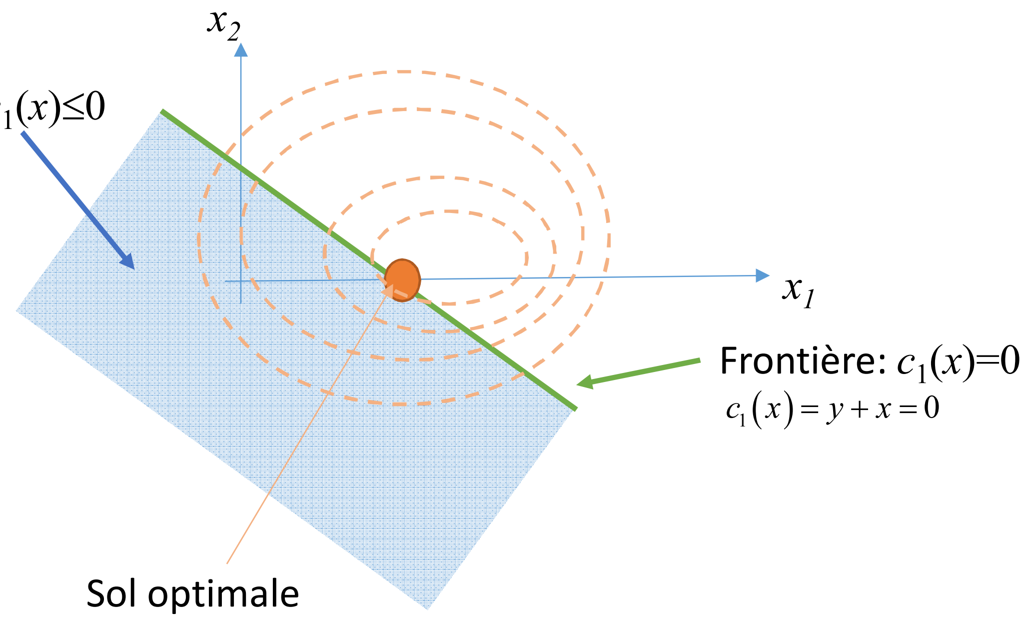
$\Rightarrow$ Le problème n'est pas contraint

$\Rightarrow$ La contrainte inégalité ne sert à rien (dans ce cas précis)

$$\left| \begin{array}{l} \min f(x,y) \\ c_1(x) = y + x \le 0 \Leftrightarrow y \le -x \end{array} \right. \qquad \Leftrightarrow \qquad \min f(x,y)$$

Sequential Quadratic Programming – *Active set method*

<u>Exemple de contrainte inégalité:</u> $c_1(x) = y + x \le 0 \Leftrightarrow y \le -x$



<u>Cas n°1 :</u> solution sur la contrainte

Comme la solution est sur la contrainte, on cherche:

$$c(x) = 0 \qquad \mu > 0$$

$$L(x, \mu, \lambda) = f(x) + \mu^T c(x)$$

$\Rightarrow$ Conditions d'optimalité similaires à un problème avec contrainte égalité !

$\Rightarrow$ On sait résoudre un pb avec des contraintes égalité !

$$\left| \begin{array}{l} \min f(x, y) \\ c_1(x) = y + x \le 0 \Leftrightarrow y \le -x \end{array} \right. \quad \Leftrightarrow \quad \left| \begin{array}{l} \min f(x, y) \\ c_1(x) = y + x = 0 \Leftrightarrow y = -x \end{array} \right.$$

Sequential Quadratic Programming – *Active set method*

En résumé:

Une contrainte inégalité est :

- soit « inutile » : ne modifie pas la solution non contrainte
- soit assimilable à une contrainte égalité

Problème:

Lorsqu'il y a plusieurs contraintes, on ne connait pas à l'avance l'ensemble des contraintes actives (Active set)

Solution:

On va essayer de deviner l'ensemble des contraintes actives au fur et à mesure de l'algorithme

=> Algorithme « SQP – active set »

Que faut il savoir faire pour estimer l'ensemble des contraintes actives ?

- Pb n°1 : Supprimer les contraintes qui ne respectent pas les conditions d'optimalité (multiplicateurs négatifs)
- Pb n°2 : Intégrer les contraintes dans cet ensemble lorsque la solution « tape » dedans

Sequential Quadratic Programming – *Active set method*

- Pb n°1 : Supprimer les contraintes qui ne respectent pas les conditions d'optimalité (multiplicateurs négatifs)

conditions d'optimalités KKT: $L(x,\mu,\lambda) = f(x) + \mu^T c(x) + \lambda^T h(x)$

Au premier ordre : la dérivée du Lagrangien est nulle $\nabla f(x) + (\nabla c(x))^T \mu = 0$

$$\Leftrightarrow \mu = -\left((\nabla c(x))(\nabla c(x))^T\right)^{-1} (\nabla c(x)) \nabla f(x)$$

Pour une solution x^k supposée optimale, on peut estimer les multiplicateurs μ_i associés à la i$^{\text{ème}}$ contrainte

1) On suppose un ensemble de contraintes actives

2) On laisse l'algorithme (SQP avec contraintes égalités) converger.

   => La solution obtenue est sur une ou plusieurs des contraintes actives

3) On calcule les multiplicateurs μ_i associés aux contraintes

   Si tous les $\mu_i > 0$ alors la solution est optimale pour les contraintes considérées

   Les $\mu_i \leq 0$ indiquent les contraintes inactives (qui doivent donc être enlever de l'ensemble des contraintes actives).

Sequential Quadratic Programming – *contraintes égalités & inégalités*

Exemple:

$$\left| \begin{array}{l} J = \min f(x,y) = x^2 + y^2 \\ h_1(x,y) = -x + x^{\min} \le 0 \Leftrightarrow x \ge x^{\min} \\ h_2(x,y) = y + 0.5 \le 0 \Leftrightarrow y \le -0.5 \end{array} \right.$$

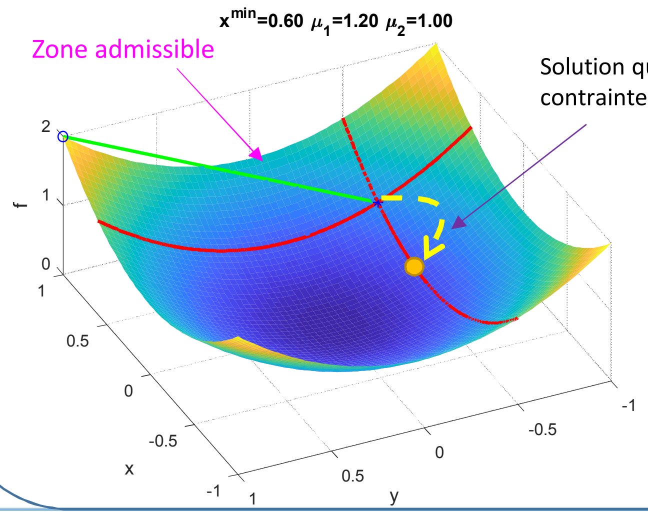
$\longrightarrow$ Sol non contrainte $\left(x_{nc}^*, y_{nc}^*\right) = (0,0)$

Solution initiale: $x^0 = (1,1)$

On applique l'algorithme SQP – égalité en supposant les 2 contraintes actives. On fait varier x^{max}.

x^{min} = 0.6 $\mu_1 > 0$: c_1 doit rester active

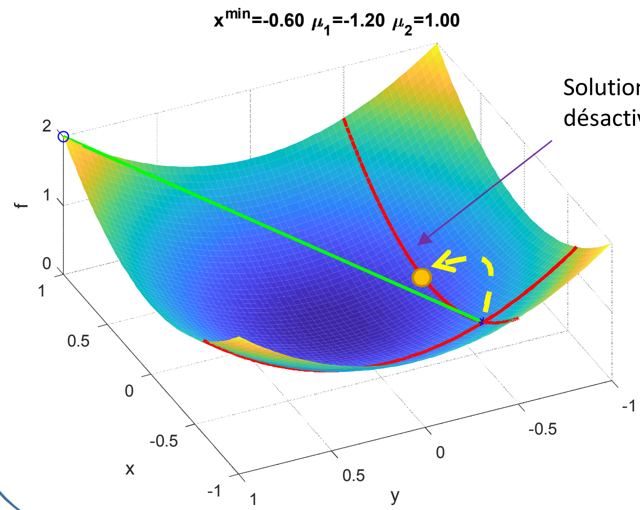
Aucune solution respectant les contraintes ne peut être trouvée.
Si on désactive la contrainte, les sol obtenus ne seront plus admissibles



Solution que l'on obtiendrait en désactivant la contrainte

x^{min} = -0.6 $\mu_1 < 0$: c_1 doit être désactivé

Une meilleure solution peut être obtenue si la contrainte est désactivée.



Solution (meilleure) que l'on obtiendrait en désactivant la contrainte

Sequential Quadratic Programming – *Active set method*

- Pb n°2 : détecter si on « tape » dans une ou plusieurs contrainte non active

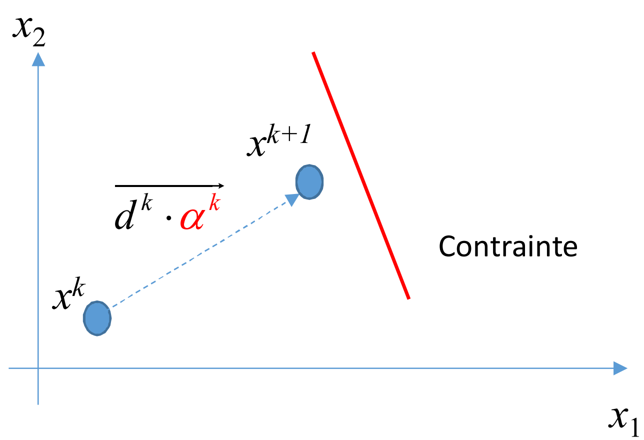
On modifie l'itération de Newton: $x^{k+1} = x^k + d^k \cdot \alpha^k$ $\alpha^k \in \mathbb{R}^+$

α^k est un scalaire qui :

- vaut 1 si aucune nouvelle contrainte n'est rencontrée
- Est plus petit que 1 si une contrainte doit être rajoutée dans l'ensemble des contraintes actives:

Cas n°1 : la solution ne rencontre aucune contrainte

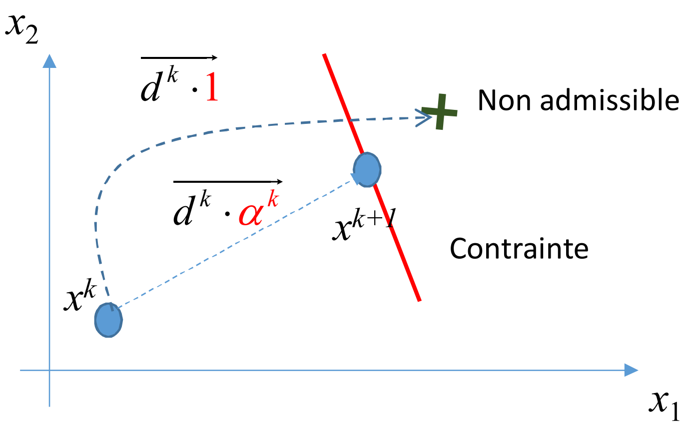
=> La contrainte reste considérée comme *inactive*



Cas n°2 : la solution rencontre une contrainte

Le pas complet (α=1) amène le point suivant dans la zone non admissible.

=> On rajoute la contraintes dans l'ensemble des contraintes actives



Sequential Quadratic Programming – *Active set method*

- Pb n°2 : détecter si on « tape » dans une ou plusieurs contrainte non active

On modifie l'itération de Newton: $x^{k+1} = x^k + d^k \cdot \alpha^k$ $\alpha^k \in \mathbb{R}^+$

α^k est un scalaire qui :

- vaut 1 si aucune nouvelle contrainte n'est rencontrée
- Est plus petit que 1 si une contrainte doit être rajoutée dans l'ensemble des contraintes actives:

Pour des contraintes linéaires, il existe une solution analytique:

Contraintes: $c_i(x) = a_i^T x - b_i \Leftrightarrow a_i^T x \leq b_i$

Pas α_i qui amène la solution sur la i^{ème} contrainte:

$$c_i\left(x^{k+1}\right) = 0 \Leftrightarrow a_i^T\left(x^k + d^k \cdot \alpha_i\right) - b_i = 0$$
$$\Leftrightarrow a_i^T x^k + a_i^T d^k \cdot \alpha_i - b_i = 0$$
$$\Leftrightarrow a_i^T d^k \cdot \alpha_i = b_i - a_i^T x^k$$

On garde un pas positif (pour ne pas reculer):

$$\Leftrightarrow \alpha_i = \max\left(0, \frac{b_i - a_i^T x^k}{a_i^T d^k}\right)$$

On garde le plus petit pas:

$$\Leftrightarrow \alpha^k = \min_i \alpha_i$$

Sequential Quadratic Programming – *Active set method*

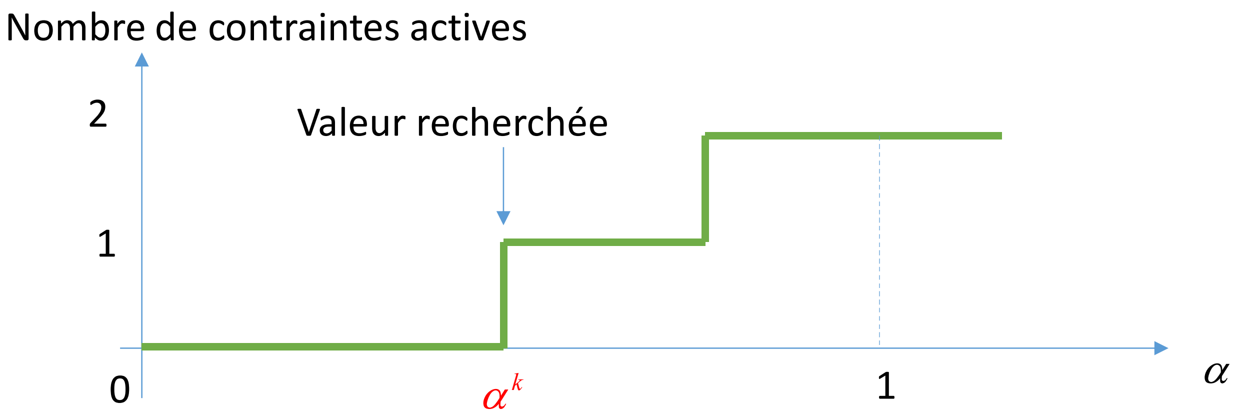
- Pb n°2 : détecter si on « tape » dans une ou plusieurs contrainte non active

On modifie l'itération de Newton: $x^{k+1} = x^k + d^k \cdot \alpha^k$ $\alpha^k \in \mathbb{R}^+$

α^k est un scalaire qui :

- vaut 1 si aucune nouvelle contrainte n'est rencontrée
- Est plus petit que 1 si une contrainte doit être rajoutée dans l'ensemble des contraintes actives:

Pour des contraintes non linéaires, il existe n'existe pas de solution simple.
Un « bricolage », peu efficace d'un point de vue numérique :
$\Rightarrow$ Utiliser une bisection (dichotomie) pour trouver la valeur de α^k telle que la première contrainte soit active
$\Rightarrow$ Garder la dernière valeur à gauche de la 1ère marche d'escalier : plus grande valeur de α^k *sans aucune contrainte active*



Sequential Quadratic Programming – *Active set method*

<u>Paramètres :</u>

Solution initiale: x^0 Tolérance contrainte $\varepsilon_c > 0$

Tolérance sur l'optimalité au premier ordre $\varepsilon_{dx} > 0$

<u>Algorithme</u>

1) $k \leftarrow 0$

$$\mu^0 \leftarrow -\left(\left(\nabla c\left(x^0\right)\right)\left(\nabla c\left(x^0\right)\right)^T\right)^{-1}\left(\nabla c\left(x^0\right)\right)\nabla f\left(x^0\right)$$

Ensemble de contraintes actives: $I^k = \left\{ i \mid c_i\left(x^k\right) > \varepsilon_c \right\}$

2) Résolution du problème SQP avec contraintes égalités: $\left| \begin{array}{l} \min f(x) \\ c_i(x) \leq 0 \quad i \in I^k \end{array} \right.$

Contraintes actives notées: $\bar{c}(x)$

Lagrangien contraintes actives noté: $\bar{L}(x,\mu) = f(x) + \sum_{i \in I^k} \mu_i^k c_i(x)$

Sequential Quadratic Programming – *Active set method*

2) Calculer (d_x^k, d_λ^k) solution de $$\begin{pmatrix} \nabla^2 f(x^k) + \sum_{i\in I} \nabla^2 c_i(x^k)\mu_i^k & \left(\nabla \bar{c}(x^k)\right)^T \\ \nabla \bar{c}(x^k) & 0 \end{pmatrix} \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} = -\begin{pmatrix} \nabla_x \bar{L}(x,\lambda) \\ \bar{c}(x^k) \end{pmatrix}$$

$$\begin{pmatrix} x^{k+1} \\ \mu^{k+1} \end{pmatrix} = \begin{pmatrix} d_x^k \\ d_\lambda^k \end{pmatrix} + \begin{pmatrix} x^k \\ \mu^k \end{pmatrix}$$

3) Calculer le plus grand pas α^k tel que x^{k+1} soit admissible, c'est-à-dire que $c(x^{k+1}) \leq 0$

On note z le n° de la contrainte rencontré si $\alpha^k < 1$

4) **SI** $\|d_x^k\| < \varepsilon_{dx}$ **ALORS**

=> l'algorithme à convergé vers un minimum local en considérant les contraintes I

=> On vérifie s'il ne faut pas supprimer des contraintes de l'ensemble I

5) $$\mu^k \leftarrow -\left(\left(\nabla c(x^k)\right)\left(\nabla c(x^k)\right)^T\right)^{-1}\left(\nabla c(x^k)\right)\nabla f(x^k)$$

6) **SI** $\mu_i^k \geq 0 \quad i = 1..p$ **ALORS**

Les conditions KKT sont respectées, solution trouvée => fin.

Sequential Quadratic Programming – *Active set method*

SINON

7) Une ou plusieurs contraintes doivent être retiré de l'ensemble I^k

=> On en retire qu'une, celle qui pose le plus de problème

$$\mu_{j0}^k \leftarrow \min\left\{\mu_j^k \mid \mu_j^k < 0\right\}$$

$$I^k \leftarrow I^k \setminus \{j_0\}$$

GOTO step 2 (la solution courante est rejetée et recalculée)

Fin SI

SINON

$\left\|d_x^k\right\| > \varepsilon_{dx}$: l'algorithme n'a pas encore convergé vers un minimum local

8) **SI** $\alpha^k = 1$ **ALORS**

9) Aucune nouvelle contrainte n'a été rencontré => on continue

$k \leftarrow k+1$ GOTO Step 2

Sequential Quadratic Programming – *Active set method*

SINON

10) Une contrainte a été rencontré car $\alpha^k < 1$
On rajoute la contrainte n°z (cf. step 3) à l'ensemble I^{k+1}

$$I^{k+1} \leftarrow I^k \cup \{z\}$$

$k \leftarrow k+1$ GOTO Step 2

Fin SI

Fin SI

Sequential Quadratic Programming – *Active set method*

Code Matlab: Attention mise en œuvre très « naive » de l'algorithme.

Utilisation de la toolbox « symbolic » pour calculer les gradients, jacobiennes et hessien

```
clear all
close all
clc

% Symbolic functions for inequalities to equalities plus slack variable
syms x y z l1 l2 real

% Critère
f(x,y)= (1-x).^2+100*(y-x^2)^2;

% Contrainte inégalité c<=0
c(1,:)=-x-0.5; % x<0.5
c(2,:)=y+0.2; % y<-0.2
p=length(c);


Var=[x;y];
df=gradient(f,Var);
d2f=hessian(f,Var);
Jc=jacobian(c,Var);

% Lagrangien
mu=sym('mu',[p,1]);
L=f+c'*mu;

dLdx=gradient(L,Var);
d2Ldx2=hessian(L,Var);
```

```
matlabFunction(f,'File','f','Vars',{Var});
matlabFunction(c,'File','c','Vars',{Var});
matlabFunction(df,'File','df','Vars',{Var});
matlabFunction(d2f,'File','d2f','Vars',{Var});
matlabFunction(L,'File','L','Vars',{Var,mu});
matlabFunction(dLdx,'File','dLdx','Vars',{Var,mu});
matlabFunction(d2Ldx2,'File','d2Ldx2','Vars',{Var,mu});
matlabFunction(Jc,'File','Jc','Vars',{Var});
```

Sequential Quadratic Programming – *Active set method*

Code Matlab: Attention mise en œuvre très « naive » de l'algorithme.

Gestion des contraintes actives: Variable `ActiveSet` : tableau de booléen indiquant si les contraintes sont actives ou non

```
clear all;
close all;
clc;


% Condition initiale
x0=[0.5;-1];

% Zone d'intéret pour les graphiques
xmin=-1; xmax=1; ymin=-1; ymax=1;

% Affichage de la fonction
X=linspace(xmin,xmax,61);
Y=linspace(ymin,ymax,62);
Z=zeros(length(Y),length(X));
Z2=Z;
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f([X(i);Y(j)]);
    end
end
```

```
figure(10);
surf(X,Y,Z,'EdgeColor','none');
hold on;
xlabel('x');
ylabel('y');
hold on;
plot3(x0(1),x0(2),f(x0),'bo');
xlabel('x');
ylabel('y');
zlabel('g');
view(-115.1,58);

% Criteres d'arret
TolOpti=1e-9;
IterMax=100;
TolContrainte=1e-10;


n=size(x0,1);
p=size(c(x0),1);

% Estimation des paramètres de Lagrange initiaux
gf=df(x0);
gc=Jc(x0);
mu0=-(gc*gc')^-1*gc*gf;

% Algorithme SQP active-set
k=1;
x=x0;
mu=mu0;
ended=0;
```

Code Matlab: Attention mise en œuvre très « naive » de l'algorithme.

```
% Initial guess
ActiveSet=c(x(:,k))>TolContrainte;

while ended==0
    fprintf('Iter : %i\n',k);
    xk=x(:,k);
    cxk=c(x(:,k));

    pactive=sum(ActiveSet);
    fprintf('  %i contraintes actives : ',pactive);
    for i=1:p;fprintf('%i ',ActiveSet(i)),end, fprintf('\n');

    Lxk=f(xk)+mu(ActiveSet,k)'*cxk(ActiveSet);
    Jcxk=Jc(x(:,k));
    Jcxk=Jcxk(ActiveSet,:);
    dfx=df(x(:,k));

    %Résolution du SQP avec contraintes égalité (active set)
    H=d2Ldx2(x(:,k),mu(:,k).*ActiveSet);
    dLx=dLdx(x(:,k),mu(:,k).*ActiveSet);

    A=[H     Jcxk';
       Jcxk  zeros(pactive,pactive)];

    B=-[dLx;cxk(ActiveSet)];
    Phi=A\B;
    dx(:,1)=Phi(1:n);
    dmu=zeros(p,1);
    dmu(ActiveSet,1)=Phi(n+1:end);

    [alpha(k),ActiveConstraints]=Bisection(x(:,k),dx,@c,TolContrainte);
    fprintf('   alpha : %.2e\n',alpha(k));

x(:,k+1)=x(:,k)+dx*alpha(k);
mu(:,k+1)=zeros(p,1);
mu(:,k+1)=mu(:,k)+dmu;

xkp1=x(:,k+1);
fxkp1=f(x(:,k+1))+mu(:,k)'*c(x(:,k+1));

figure(10)
plot3(xkp1(1),xkp1(2),f(xkp1),'r*');
plot3([xkp1(1) xk(1)],[xkp1(2) xk(2)],[f(xkp1) Lxk],'g');
```

Sequential Quadratic Programming – *Active set method*

Code Matlab: Attention mise en œuvre très « naive » de l'algorithme.

```
if norm(dx)<TolOpti
    % Estimation des paramètres de Lagrange de toutes les
    % contraintes
    gf=df(x(:,k+1));
    gc=Jc(x(:,k+1));
    mu2=-(gc*gc')^-1*gc*gf;
    cxkp1=c(x(:,k+1));
    fprintf('  mu2: ');
    for i=1:p;fprintf('%.2e ',mu(i,k+1)),end
    fprintf('\n');

    oldActiveSet=ActiveSet;
    if all(mu2>0)
        ended=1;
        StepAcceped=1;
    else
        ended=0;
        StepAcceped=0;
        % Update active set : remove unnecessary constraint
        [mu_min]=min(mu2(oldActiveSet));
        ActiveSet=ActiveSet&(mu2~=mu_min);
    end
else
    StepAcceped=1;
    % Not converged yet
    if alpha(k)<1
        % At least one constraint is active
        ActiveSet=ActiveSet|ActiveConstraints;
    end
end


fprintf('  mu: ');
for i=1:p;fprintf('%.2e ',mu(i,k+1)),end
fprintf('\n');
```

```
% Affichage
    xkp1=x(:,k+1);
    fxkp1=f(x(:,k+1))+mu(:,k)'*c(x(:,k+1));
    ended=(k>IterMax)||ended;

    if ~StepAcceped
        x(:,k+1)=x(:,k);
        mu(:,k+1)=mu(:,k);
    end
    k=k+1;
end

fprintf('Valeur finale : x=[%.2f,%.2f] \n',x(1,k),x(2,k));
fprintf('Nombre d''iterations : %i\n',k)
```

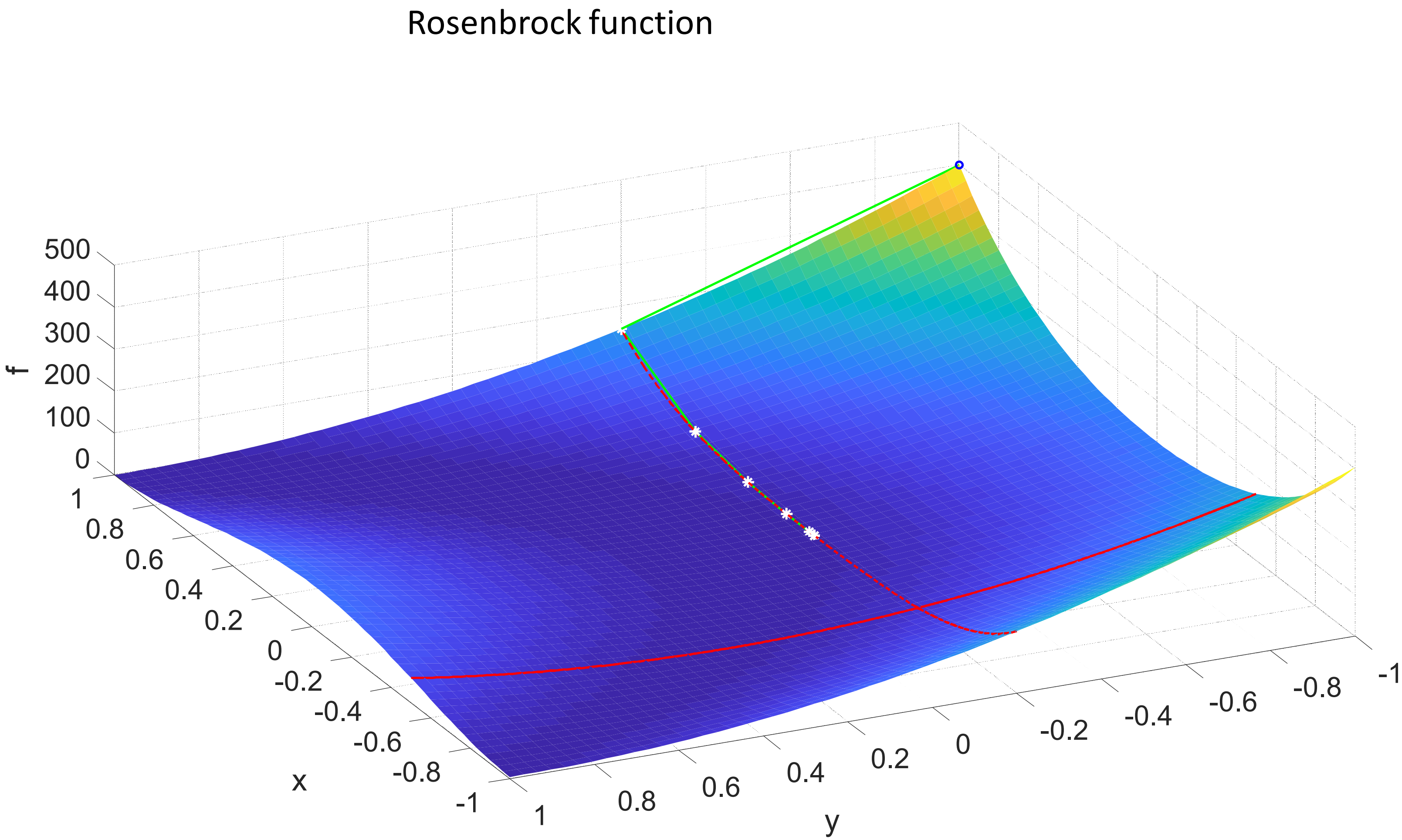
Sequential Quadratic Programming – *Active set method*

Code Matlab: Attention mise en œuvre très « naive » de l'algorithme.

```
function [alpha,ActiveSet]=Bisection(xk,dx,c,TolContrainte)
% Renvoie un pas alpha admissible et la liste des contrainte qui
% ont empéché un pas plus grand-
AlphaMin=0;
AlphaMax=1;
% Try full step alpha=1
crit_alpha_max=sum(c(xk+dx)>TolContrainte);
ActiveSet=c(xk+dx)>TolContrainte;
if crit_alpha_max==0
    alpha=1;
    return;
end
% We must reduce the step
ended=0;
TolAlpha=1e-8;
i=1;
while ended==0
    alpha=(AlphaMax+AlphaMin)*0.5;
    crit=sum(c(xk+dx*alpha)>TolContrainte);
    if crit>0
        AlphaMax=alpha;
        ActiveSet=c(xk+dx*alpha)>TolContrainte;
    else
        AlphaMin=alpha;
    end
    ended=((AlphaMax-AlphaMin)<TolAlpha)|(i>1000);
end
end
```

Sequential Quadratic Programming – *Active set method*

$$\begin{cases} \min f(x,y) = (1-x)^2 + 100(y-x^2)^2 \\ c(x,y) = \begin{pmatrix} -x-0.5 \\ y+0.2 \end{pmatrix} \le 0 \end{cases}$$



$$(x^*, y^*) = (0.02, \text{-}0.2)$$

SQP active set

9 itérations boucle principale